Chapter

COMPLEMENTING THE ROLE MODEL METHOD WITH PETRI NET TECHNIQUES IN STUDYING ISSUES OF DATA FRESHNESS OF THE FOUR-SLOT MECHANISM

Fei Xia and Ian Clark

Department of Computing Science, University of Newcastle, and Department of Electronic of Engineering, King's College London.

fei.xia@ncl.ac.uk, ian.clark@kcl.ac.uk

Key words: Petri nets, role models, asynchronous communication mechanisms

Abstract: Simpson's role model method [7, 8] was designed for the analysis of synchronisation-free data communication mechanisms employing shared memory and has been shown to be especially useful for the representation and analyses of data freshness properties. Previously published analyses using the role model method have employed proprietary state space search techniques developed by Simpson. Here a formal definition of role models is given and a way of representing role models using Petri nets is presented. Potential advantages of analysing systems using the role model method complemented with Petri net techniques are demonstrated with a case study of data freshness properties of a data communication algorithm.

1. INTRODUCTION

The use of fully asynchronous processes is advantageous in many hard real-time distributed computer systems. For instance, the complete elimination of time interference in data communications between concurrent processes makes it possible to accurately predict the temporal progress of each process in the system because the timing of each one is completely independent. In certain safety critical systems it may also be required that a process cannot be temporally connected to any other processes and must progress on its own pace. In the real time software design method MASCOT, the pool type IDA accommodates the possibility of a complete lack of synchronisation between the reading and writing processes [1, 2].

The class of data communication mechanisms described and studied by Simpson [3] are designed for transferring data between completely synchronisation-free processes. The mechanisms employ from one to four data areas (called *slots*), implied to be in memory shared between the writer and the reader. Some of the mechanisms have logic in the form of control variables to steer the writer and reader to prevent them from accessing the same slot simultaneously and to cause the reader to obtain the most up-to-date data provided by the writer. The reader and writer are not subject to any relative timing constraints and thus are completely temporally independent without the need for synchronisation, wait states or arbitration in the conventional, request/grant, sense. The general organisation of these types of mechanisms is shown in Figure 1. They will be referred to as *slot mechanisms* in this report.

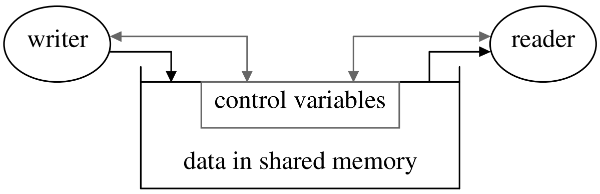

Figure 1. **Scheme of slot mechanisms.**

It is vital, especially in view of the lack of synchronisation, for the slot mechanisms to be proved to pass on data that is both coherent and fresh. Although other solutions to the problem exist [4, 5], Simpson was the first to show a solution using four slots arranged in two pairs to maintain data coherence and data freshness, having found that up to three slots arranged linearly cannot maintain data coherence [3, 6]. In order to prove the data coherence and data freshness properties of the slot mechanisms, Simpson introduced the role model method [7, 8], which is a novel notation describing discrete state transitions to assist a process of exhaustive reasoning and state space search. A proprietary technique, employing what are called *transition diagrams*, which is similar to reachability analysis in Petri nets [9], was proposed by Simpson for the analyses of role models [7].

As a result of the novel nature of the role model method there have been no supporting studies using it for analysis published by other researchers. To date, there has been no attempt to formally define role models. In addition, although it has been claimed that automated, computerised analyses have been carried out based on transition diagrams [8], no software packages by other sources exist by which any analysis result using Simpson's software may be verified.

It has been shown that Petri net models of the slot mechanisms can be obtained and that the loss of data coherence, as it is signified by a relatively simple state (that of the reader and writer accessing the same slot at the same time), can be checked for with the help of the Petri net models in a straight forward manner [10, 11]. However, as data freshness is affected not only by simple states but also by the trajectories of states, the role model method is more useful for its analysis [10, 12, 13].

In this report, the role model method is defined in a formal manner. A method of deriving Petri net representations for role models is then given. The intention is to replace or supplement the transition diagram analysis technique with the analysis techniques available for Petri nets to realise more automated and verifiable analyses of asynchronous communication systems. An example is provided where the data freshness properties of a four-slot mechanism algorithm is investigated.

This report should be read in conjunction with [3], [7] and [8].

2. ROLE MODELS

The role model method was "developed specifically to deal with the [slot mechanisms] in which small shared control variables are used to co-ordinate access to shared data" [8].

Definition 1. **Role model basics**

A role model is a tuple, $M = (A, E, R, S, s_0)$.

$A = \{a_1, a_2, \ldots, a_n\}$ is a finite set of *agents* (implied but not formally described in [7]), $n \geq 0$. $R = \{r_1, r_2, \ldots, r_m\}$ is a finite set of *roles*, $m \geq 0$. $E = \{e_1, e_2, \ldots, e_l\}$ is a finite set of *events*, $l \geq 0$. The sets of agents, roles and events are disjoint with one another, $A \cap R = \varnothing$, $A \cap E = \varnothing$, $E \cap R = \varnothing$.

An agent may *assume* a number of roles (known as the *role pattern* of the agent [7]). An event may or may not be permitted to *occur*. A *state*, $s_i \in S$, is a complete description of the current role pattern of each agent, in the form of a set of role patterns. Such sets of patterns are generally known as *pattern expressions* or in short *expressions*. In addition, a state also includes an implied specification of which events may occur next.

An event's occurrence may *modify* a state to produce another state. An event e_j, $j \in [1, l]$, is therefore a *state transition function*, a mapping from states to states, $e_j \in E: S \times S$, and is the only provision by which states may be changed.

Events occur according to interleaving semantics, in other words, at most one event occurs at any time, because “events are regarded as distinct” [8].

The dynamic progression of the model consists of the occurrences of events from the *initial state* $s_0 \in S$.

Definition 2. **Role patterns**

A role pattern, $p_R \subseteq 2^R$, is a subset or a group of subsets of R. A simple way to express it is by writing the names of the *member roles* together in a continuous string. For instance, $r = r_j r_k$ is a role pattern, if r_j and r_k are role names and $r_j \neq r_k$. Agent a_i assuming the roles of r_j and r_k may be written as $a_i r_j r_k$ or $a_i r$ if $r = r_j r_k$.

In addition to role names, the *pattern operators* “*”, “0” and “$^-$” may appear in role patterns.

Definition 3. **Expressions**

A pattern expression (or expression in short) is a set of patterns. Expressions are written in the conventional list style, i.e. member patterns separated by commas.

There is at most one role pattern for each agent of the model in any expression. The simplest non-empty expression is a single pattern consisting of one member role or one pattern operator.

An expression and/or some of its member patterns may be enclosed in (), [], or { }. In general, square brackets indicate that the expression describes a subset of all possible patterns and curly braces indicate that the expression describes all possible patterns within its scope. These and parentheses are used to discriminate between parts of expressions.

The following are example expressions:

$[a_1 0, (a_2 r_4 r_5, a_3{+}r_4, a_4{*}), a_5{+}r_1, {+}e_3 e_1 e_2]$

$[a_1 r_4 r_5{*}, \{a_2{*}, a_3{*}, a_4{*}\}, a_5 0, e_1 e_3]$

$\{*\}$

where a_i, $i \in [1,5]$, are agents, r_j, $j \in \{1,4,5\}$, are roles, and e_k, $k \in [1,3]$, are events. The parentheses and braces indicate that agents a_2, a_3 and a_4 may represent something distinctive, or are being highlighted for some reason.

Definition 4. **Transition statements**

A transition statement consists of an *input expression* followed by an $\rightarrow$ followed by an *output expression*. It describes an *action*, which, when *carried out*, results in a *modification* of the patterns of the input expression as specified by the output expression.

A role member in the role pattern for an agent in an input expression indicates that the agent assumes this role before the transition statement. An “*” in an input expression indicates irrelevance for the transition statement (don't care). A "0" in an input expression indicates that the agent assumes no roles.

A role name in the role pattern for an agent in an output expression indicates the modification of "adding if previously not assumed", i.e. the specified role is added to the role pattern of the agent if it is not there previously. If the agent already assumes this role no change is implemented.

An "*" in an output expression indicates no change.

A "¯" on top of a role name in an output expression indicates the modification of "removing if present", i.e. if the agent previously assumed the role after the statement it will not assume it any more.

Therefore the basic transition statements, which specify the simplest modifications, are as follows:

$$[ar''] \rightarrow [ar'] \tag{1}$$

$$[ar''] \rightarrow [a\bar{r}] \tag{2}$$

$$[ar''] \rightarrow [a*] \tag{3}$$

$$[ar''] \rightarrow [a0] \tag{4}$$

The transition statement (1) describes an action of adding the role *r*' to the role pattern of agent *a*, without modifying anything else. After this the agent assumes the role *r*' while retaining any other role it had. The transition statement (2) describes an action of removing the role *r* from the role pattern of agent *a*, without modifying anything else. After this the agent *a* does not assume the role *r*, while retaining any other role it had. The transition statement (3) describes an action of no change to the role pattern of the agent *a*. After this the agent *a* retains all the roles it had. The transition statement (4) describes an action of removing all roles from the role pattern of the agent *a*. After this the agent *a* assumes no roles.

Definition 5. **Interleaving semantics of transition statements**

A transition statement must be constructed in such a way that its overall modification is obtained by carrying out each of the basic transition statements it contains according to interleaving semantics (one at a time) in arbitrary order. In practice, this requires that within any transition statement the same role is not specified to be modified in conflicting ways and effects that depend on the order of actions must be specified using multiple transition statements.

Definition 6. **Pattern compatibility**

Two patterns p_x and p_y are said to be compatible with each other iff no role is specified present or absent in conflicting ways in p_x and p_y. This relation is denoted by the symbol '~'. The complement relation, incompatibility, is denoted by '/~'.

For instance, $a^* \sim ar$ for any r.

Definition 7. **Event functions**

An event e_i is a state transition function, $e_i \in E$: $S \times S$ which describes the state change when e_i occurs. It consists of p transition statements, p being a finite number, $p \geq 1$. The description of any event e_i is written as

e_i: $I_{ei1} \rightarrow O_{ei1}$, $I_{ei2} \rightarrow O_{ei2}$, …, $I_{eip} \rightarrow O_{eip}$

where I_{eij} is the jth input and O_{eij} is the jth output expression of e_i, $j \in [1, p]$.

The effect of an event's occurrence is obtained by carrying out the transition statements of the event function one at a time in the sequence specified.

Definition 8. **The organisation of events in systems**

Events are regarded as atomic, i.e. no event may start when another is occurring. Events may be organised into sequential *processes* in the fashion of conventional programming languages, i.e. by writing them one after another in a sequence. Such sequential processes may also be specified to be concurrent to one another by writing them separately. When more than one events belonging to different concurrent processes are ready to occur at any state, which one occurs next is not deterministic.

3. PETRI NET REPRESENTATION OF ROLE MODELS

Since a role model represents a discrete system with a bounded number of potential states (A, R and E being bounded for any known model) whose actions follow interleaving semantics, a subset of what classical Petri nets can represent [Peterson 1981], it should be possible to find an equivalent Petri net for any role model. Here techniques are developed whereby a safe (1-safe) Petri net representation of any given role model may be found.

Proposition 1. **Role patterns**

It is proposed to represent the notion of an agent and its possible role patterns with 1-safe places. An agent assuming a particular role in a pattern is represented by a particular place having the marking 1. An agent not assuming a particular role in a pattern is represented by its corresponding place having the marking 0. Complementary places are employed to avoid the need for inhibitor arcs. This necessitates a potential $2m$ such places for each agent in the model and a maximum of $2m \times n$ such places for all agents.

The role pattern of $a_1r_4r_5$, assuming that m=5, is thus represented by the Petri net fragment and marking in *Figure 2*. The role pattern of a_10 is represented by marking all the (not) places and unmarking all the other places in *Figure 2*.

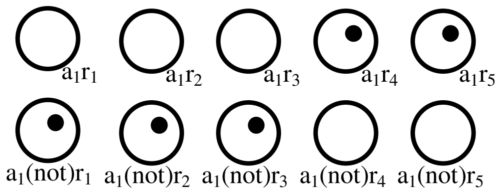

Figure 2. **Petri net representation of $a_1r_4r_5$.**

Rules are provided below for the representation of statements (1) ~ (4). Markings in the figures of this section generally represent the terminal condition of the transition statements (i.e. after the modifications have been carried out).

Proposition 2. **Petri net representation of basic transition statements**

The transition statement $[ar"] \rightarrow [ar']$ is represented by the Petri net fragment in Figure 3. The transition statements (2) ~ (4) are represented by the Petri net fragments in Figure 4 ~ Figure 6.

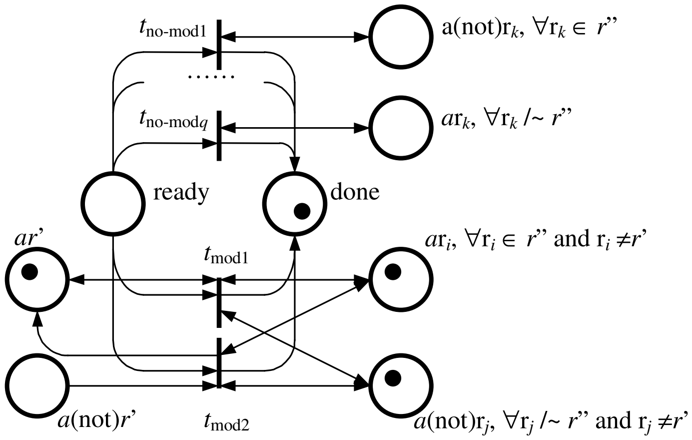

Figure 3. **Petri net representation of (1).**

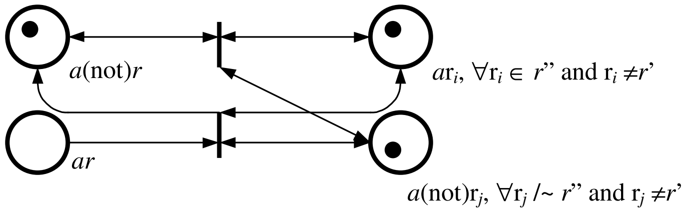

Figure 4. **Petri net representation of (2).**

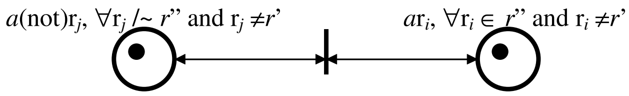

Figure 5. **Petri net representation of (3).**

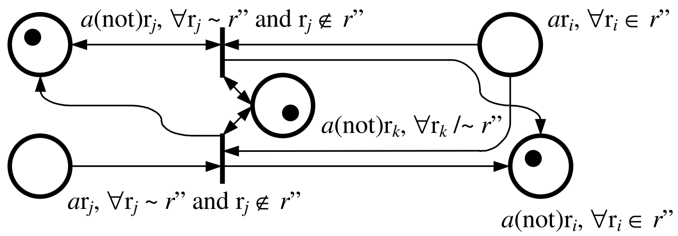

Figure 6. **Petri net representation of (4).**

There are two types of transitions in Figure 3. One type represents the proper modification according to the basic transition statement and include the transitions t_{mod1} and t_{mod2}. The roles of a other than r’ need to be referenced by these transitions, depending on if and in what way they are compatible with the pattern r”. If a role is a member of r”, the place representing it must be marked for t_{mod1} or t_{mod2} to fire. If a role is not compatible with r” the place representing its complement must be marked for t_{mod1} or t_{mod2} to fire. If a role is compatible with but not a member of r”, i.e. it is compatible with r” solely because r” contains an *, places representing it are not connected with t_{mod1} and t_{mod2} in any way. These connections represent the fact that the input expression of the transition statement must be satisfied (i.e. it is compatible with the present state) for its action to be carried out. The connections between transitions t_{mod1} and t_{mod2} and the place representing ar’ and its complementary place represent the modification of this transition statement. The markings of these places are either modified or retained depending on whether the role r’ is assumed by the agent a or not before the action.

The “ready” and “done” places are provided so that the sequential nature of transition statements within an event may be represented (below). These places are not explicitly drawn in the other Petri net fragments (Figure

4 ~ Figure 6) but are assumed to be present. One of the transitions firing in this net fragment takes a token away from place ready and deposits it in place done, and the net fragment only becomes active when ready is marked.

The number of transitions named $t_{\text{no-mod}x}$, $x \in [1, q]$ where q is a finite integer are needed to ensure that the net fragment does not introduce deadlocks artificially, i.e. if place ready is marked, at least one transition between it and place done is enabled. At least one of these "no modification" transitions is enabled if a does not assume the pattern r". These transitions are also assumed to be present in all the other net fragments in this proposition but are schematically shown in only once Figure 3.

Proposition 3. **Representing transition statements**

It is proposed that a transition statement t containing a finite number, h, of basic actions be represented with the Petri net model in Figure 7.

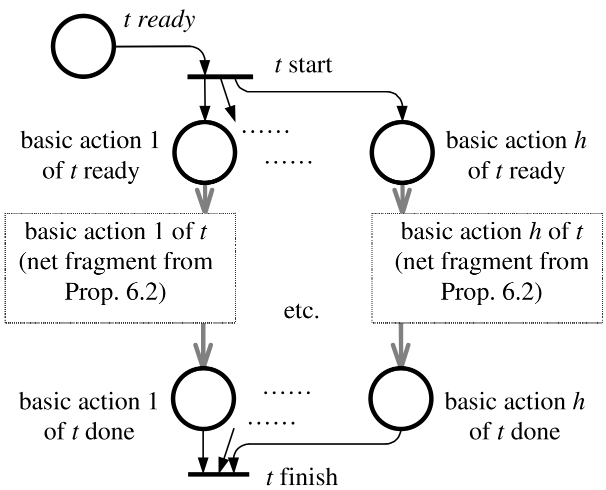

Figure 7. **Petri net representation of a transition statement.**

The structure of Figure 7 conforms with the specification that within a transition statement, the basic actions are carried out according to interleaving semantics in arbitrary order.

Proposition 4. **Representing events**

It is proposed that an event e with a finite number, i, of transition statements be represented by the Petri net model of Figure 8.

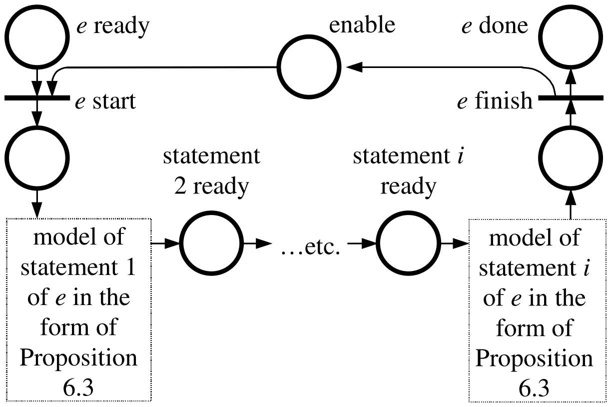

Figure 8 Petri net representation of an event.

The place enable in Figure 8 is provided so that events are atomic in the model as in the definition. This is a place shared globally among all model nets of events. The start transition of each event model has it as an input place, while it is an output place of the finish transition of each event model. This ensures that when transitions in an event model are firing, no other event model may become active until the current event's finish transition has fired. The optional event ready and done places facilitate the representation of sequential processes in the way described in [9].

4. CASE STUDY

The most recently proposed four-slot mechanism algorithm in [8] employs one fewer control variables than the one found in [7]. It also has a more streamlined organisation. This new algorithm is used here in a case study to demonstrate the finer points of the role model method and the use of Petri net techniques to supplement the analysis of role models. The algorithm itself, for a data communication mechanism of the type of Figure 1, is reproduced here in Figure 9. The algorithm includes two access procedures, one each for the writer and the reader. Each access procedure is assumed to be imbedded within a larger cycle loop in which data to be communicated is produced or consumed. Depending on the arrangement of the larger writer and reader cycles, the access may be open or limited. Taking the writer process as an example, the writer access procedure is assumed to be part of a cycle loop of "loop, produce data, write access procedure, end loop" if it has limited access. If it has open access, the statement *wr* in the access procedure would include the actions of preparation/production of data, i.e. the data is written as it is produced. In this algorithm, the reader and writer procedures are regarded as sequential within themselves, but arbitrary interleaving is allowed between the two processes. Thus it provides for complete temporal independence of the two sides and has been described as "fully asynchronous" or "synchronisation free".

Writer	Reader
wr: $d[ip, w[ip]] := input$	$r0$: $r := w$
$w0$: $ip := \overline{ip}$	$r1$: $op := \overline{ip}$
$w1$: $w[ip] := \overline{r[ip]}$	rd: $output := d[op, r[op]]$

Figure 9. **Four-slot data communication mechanism algorithm.**

Simpson applied role model based analysis on the algorithm in Figure 9 and verified its data coherence, data freshness and data sequencing properties [8]. Data coherence issues of the four-slot mechanism have been extensively investigated using Petri net based techniques [10]. In this case study, the issue of data freshness is the focus. Definitions concerning data freshness, obtained from [7] and [8], are given below.

Definition 9. **Latest and previous latest data items**

At any time, the latest item of data is found in the slot accessed by the last completed *wr* statement, and the previous latest item of data is found in the slot accessed by the last but one completed *wr* statement.

Data freshness issues are relevant only at the beginning of a reader access to a slot, i.e. at the beginning of the statement *rd* in Figure 9. Checking for data freshness obviously has to be done at this point.

Definition 10. **Data freshness**

Data freshness is maintained if the reader always accesses an acceptable slot. Acceptable slots are:

1. the slot containing the item of data which was the latest at the beginning of the pre-sequence of the current reader cycle, $slot_L$;
2. the slot containing the item of data which is the previous latest at the beginning of the pre-sequence of the current reader cycle, $slot_P$, only if the pre-sequence of this reader cycle started while a writer post sequence is in progress;
3. the slot containing the latest item of data before the beginning of the current reader access, $slot_{LL}$.

A reader cycle consists of one cycle of consecutive $r0$, $r1$ and rd statements. The statements $r0$ and $r1$ constitute the pre-sequence of the reader cycle they belong to. The post sequence of a writer cycle consists of the statements *w*0 and *w*1.

In Definition 10, 1. is intuitive, as the reader access procedure uses the pre-sequence to determine where the latest item of available data is ($slot_L$), and a successful attempt at obtaining this item of data, even in the event of newer data becoming available during the reader pre-sequence ($slot_{LL} \neq slot_L$), cannot be regarded as a data freshness failure. In comparison, 3. is meant to cover the case in which the reader is after all able to obtain the location of $slot_{LL}$, even when $slot_{LL} \neq slot_L$, by design or by accident. It is recognised in 2. that if a writer post sequence and the pre-sequence of the current reader cycle overlap in time, the reader should not be expected to obtain the location of $slot_L$ because this information may not have been completely indicated by the writer post sequence yet. However the reader should be expected to at least obtain the location of $slot_P$ in this case, as that should have been indicated by the post sequence of the previous writer cycle.

From [10], it is obvious that such information as which slot contains the latest item of data is not directly available from the Petri net models used to study data coherence of the slots mechanisms. These models have only reading, not reading, writing and not writing as the slot related states. Although a careful study of all trajectories generated by a full reachability search should reveal information concerning data freshness properties, extracting it is not a convenient and straight forward process.

More state variables are added in [7] to facilitate the study of data freshness properties using the role model method in the form of data freshness related roles. These make it possible to keep track of $slot_L$, $slot_{LL}$, and $slot_P$ during a state space analysis.

The methodology used in this investigation of data freshness properties of the four-slot mechanism is as follows:

1. Propose a formal definition of role models (not available from [7] but needed for the following step).
2. Develop general techniques whereby a Petri net representation may be found for any role model in an automatic way.
3. Find the Petri net representations of the data freshness related role models in [7] and carry out reachability searches using different software packages.

The additional roles introduced in [7] for analysing the data freshness properties of the four-slot mechanism are listed in *Table 1* below.

Table 1. **Data freshness related roles.**

Roles	Representations
W	slot is being written to
R	slot is being read from
F	slot contains latest item of data known to the writer ($slot_L$ or $slot_{LL}$)
P	slot contains previous latest item of data known to the writer ($slot_P$)
V	slot is $slot_L$ and/or $slot_{LL}$, or $slot_P$ when the pre sequence of the current reader cycle overlaps with a writer post sequence.

The Petri net models used in [10] do not have explicit representation of these roles except for *R* and *W*.

The data freshness related events and their transition statements are listed in Figure 10 below.

Writer:

$$wae: \quad [F^*, *, *, *] \rightarrow [P, *, *, *]$$

$$[W^*, *, *, *] \rightarrow [VF, \overline{F}, \overline{F}, \overline{F}]$$

$$wab: \quad [*, *, *, *] \rightarrow [\overline{P}, \overline{P}, \overline{P}, \overline{P}]$$

Reader:

$$rae: \quad [F^*, *, *, *] \rightarrow [V, \overline{V}, \overline{V}, \overline{V}]$$

$$[P^*, *, *, *] \rightarrow [V, *, *, *]$$

Figure 10. **Data freshness related events.**

The agents in these transitions statements are the slots. The event *wae* occurs at the end of a writer slot access. The event *wab* occurs at the beginning of a writer slot access. The event *rae* occurs at the end of a reader slot access. During these events the freshness related roles *F*, *P*, and *V* are updated. They should be checked to see if the reader is accessing an acceptable slot during every *rd*.

In Figure 10, the absolute positions of the slot agents are flexible, providing for a shorthand of presentation. If *x*, *y*, *z*, and *u* are role patterns, the expression [*x*, *y*, *z*, *u*] represents the situation of one of the slots assuming the *x* role pattern another one the *y* pattern, still another one the *z* pattern, and the last one the *u* pattern. It is not, in this case, indicated which of the slots 00 ~ 11 assumes which of the patterns. This type of shorthand presentation of a role model can only be used if the absolute information about each agent's role pattern is not of significance for the transition statement.

The relative positions of the role patterns are significant, however, in that they are consistent within one transition statement. In other words, once the input expression is written the output expression must correspond to it. For instance, the transition statement

$$[F^*, *, *, *] \rightarrow [V, \overline{V}, \overline{V}, \overline{V}] \quad (5)$$

Represent the action of adding the *V* role to the slot currently holding the *F* role and removing the *V* role from all other slots.

If a reader access statement *rd* is directed to a slot with the *V* role data freshness is regarded as being maintained.

Considering Definition 10, 1. is checked for with the *V* role which was updated at the end of the last reader access (the beginning of the pre-sequence of the current reader cycle) to $slot_L$. Since the writer events do not remove the *V* role from any slot and this *V* role is not removed by the other transition statement in the event *rae*, this remains set until the current *rd*. For 2. in Definition 10, the *P* role is set at the end of a writer access but removed at the beginning of the next writer access. In other words, the *P* role only exists on a slot during the writer post sequence. If the reader pre-sequence starts during such a period, the *rae* event duly assigns the *V* role to this slot, signifying the fact that the reader pre-sequence overlaps a writer post sequence hence the reading from $slot_P$ is permitted. Finally, 3. in Definition 10 is managed by giving $slot_{LL}$ the *V* role together with the *F* role at the *wae* event. This ensures that even if a writer access happened immediately before a reader access and after the reader pre-sequence, the reading from the slot just accessed by the writer would be regarded as acceptable.

Petri net models of the events in Figure 10 have been developed according to Proposition 1 ~ Proposition 4. Part of the model of transition statement (5) is shown below in Figure 11. The no-modification transitions (Figure 3) are not shown in Figure 11 but are present in the models used in the study.

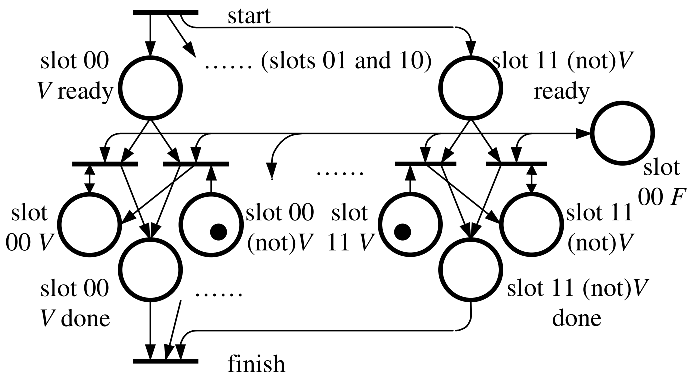

Figure 11. **Part of Petri net representation of (5).**

The Petri net model for the complete four-slot mechanism with additional provision for studying data freshness properties are constructed in the form of Figure 12. The monitoring net consists of models of the data freshness related events in Figure 10 while the original model used for data coherence analysis in [10] has been retained as the main net. The atomicity of the events in the monitoring net and the integrity of the main net are maintained with a global enable place. Whenever a transition firing in the main net signifies that an event in the monitoring net should start, the token in the enable place is passed to the appropriate part of the monitoring net while both the main net and the other parts of the monitoring net shut down. Since the monitoring net only updates the freshness related role states the main net is not affected by its operations. The monitoring net goes into action at the points within the main net shown in Figure 13. Since both *wr* and *rd* statements are regarded as non-atomic in the main net model [10], these are indeed transitions in the main net. The interface between the main and monitoring nets is schematically shown in the example of Figure 14.

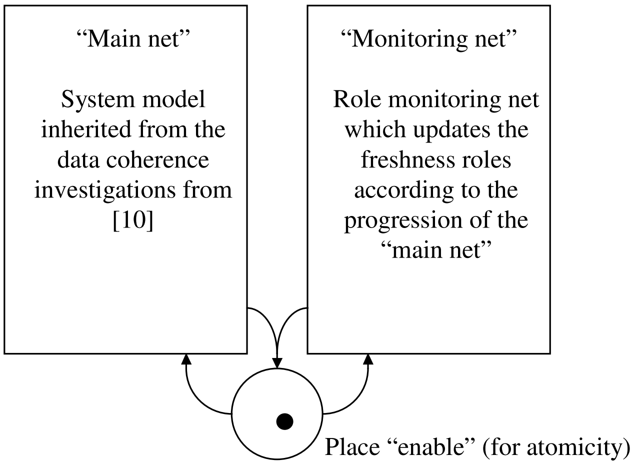

Figure 12. **Structure of Petri net model used in data freshness study.**

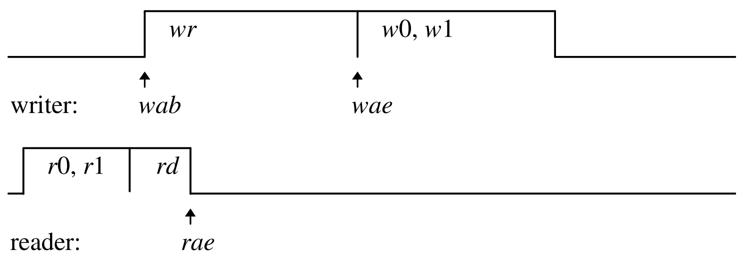

Figure 13 **Correspondence of main net statements with monitoring net events.**

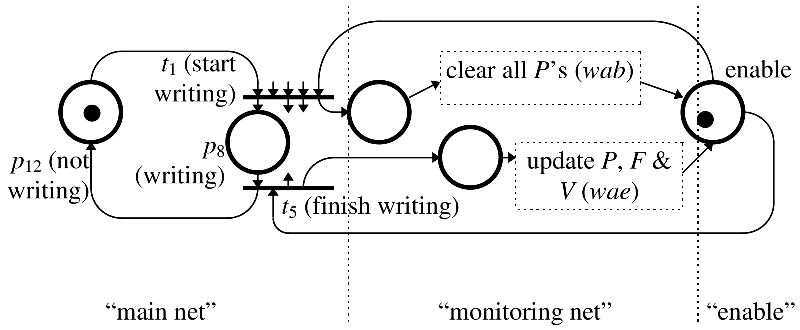

Figure 14 **Updating data freshness roles on slot *d*[0,0] for the first writing step *wr*.**

Reachability searches using the two software packages employed in the previous chapters have been run on the data freshness study model. The main net of this model is the version discussed earlier in this chapter based on the non-atomic assumptions for the control variable statements. The results confirm that no reachable marking signifies the reading of a $\overline{V}$ slot. This indicates that the four-slot mechanism maintains data freshness under normal operating conditions, and so supports the results reported in [7]. The increase of reachable states resulting from the vast number of combinations of role states on all four slots did not present a problem for either of the software packages. This indicates that they could be useful for the analysis of more complex net models.

5. CONCLUSIONS AND FUTURE WORK

Simpson's role models are presented in a formal way. It has been shown that role models have significant advantages when there is a need to track state transitions over unknown numbers of steps, for instance, when properties such as data freshness are studied.

A way of representing role models using Petri nets has been presented and a general method for using Petri net techniques to complement the role model method has been tried and proved to be viable. Certain finer points of using the role model method in checking slot mechanisms, especially their data freshness and sequencing properties are explored with a case study. Two software packages developed independently have been used to increase confidence in the analysis results obtained.

Since two different four-slot mechanism algorithms have been shown to behave differently where data sequencing is concerned, further analyses taking advantage of the availability of easy to use Petri net techniques may point out a way to improve data sequencing and related qualities of future slot mechanism algorithm designs.

From related work [14], it seems natural that coloured Petri nets (CPNs) [15] may be used to represent role models in a more straight forward manner, especially considering the shorthand notations of the latter. Therefore this may be a profitable technique to use in future investigations of the role model method. It would also be interesting to see if the role model method may be employed in the study of systems other than slot mechanisms.

This work is part of on-going studies at King's College London and the University of Newcastle into asynchronous communication algorithms and mechanisms.

6. REFERENCES

1 Joint IECCA and MUF Committee on Mascot (JIMCOM), The official handbook of Mascot, Computing Division, N building, Royal Signals and Radar Establishment, St Andrews Road, Malvern, Worcestershire, WR14 3PS, UK, June 1987. (Now available from the Defence Research Information Centre, DERA, Kentigern House, 65 Brown Street, Glasgow. G2 8EX.)

2 Simpson, H.R., "The MASCOT method", Software Eng. Journal, Vol. 1, (3), pp.103-120, 1986.

3 Simpson, H.R., "Four-slot fully asynchronous communication mechanism", IEE Proceedings, Vol. 137, Pt. E, No. 1, pp.17-30, January 1990.

4 Kirousis, L.M., "Atomic multireader register", Proc. 2nd Int. Workshop on Distributed Computing, Amsterdam, LNCS-312, pp.278-296, Springer Verlag, 1987.

5 Tromp, J., "How to construct an atomic variable, Proc. 3rd Int. Workshop on Distributed Algorithms, Nice, LNCS, Springer Verlag, pp.292-302, 1989.

6 Simpson, H.R., "Fully-asynchronous communication", IEE Colloquium on MASCOT in real-time systems, pp.2/1-2/6, May 12, 1987.

7 Simpson, H.R., "Correctness analysis of class of asynchronous communication mechanisms", IEE Proceedings, Vol. 139, Pt. E, No. 1, pp.35-49, January 1992.

8 Simpson, H.R., “New algorithms for asynchronous communication” and “Role model analysis of an asynchronous communication mechanism”, IEE Proceedings on Computing and Digital Techniques, Vol. 144, No. 4, pp.227-240, July 1997.

9 Peterson, J.L., Petri net theory and the modeling of systems, Prentice-Hall, 1981.

10 Xia, F, Clark, I.G. and Davies, A.C., “Petri-net based investigation of synchronisation-free interprocess communication in shared-memory real-time systems”, Proc. 2nd UK Asynchronous Forum, University of Newcastle upon Tyne, Newcastle upon Tyne, UK, July 1-2, 1997.

11 Semenov, A. and Yakovlev, A., “Contextual net unfolding and asynchronous system verification”, Technical report, No.572, Department of Computing Science, University of Newcastle, April, 1997.

12 Clark, I.G., PhD thesis, King’s College, University of London, in preparation.

13 Xia, F., Combining MASCOT with Petri nets, PhD thesis, King’s College, University of London, to be published in 1998.

14 Clark, I.G., Xia, F., Yakovlev, A.V. and Davies, A.C., “Petri net models of latch metastability”, Electronics Letters, pp.635-636, Vol.34, No.7, April 2, 1998.

15 Jensen, K., Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts. Monographs in Theoretical Computer Science, Springer-Verlag, 1992.